Grid Computing for Load Balancing Strategies

Kavya S.A[1], M.V.Panduranga Rao[2], S.Basavaraj Patil[3]

[1]Dept of Computer Science and Engineering
BTL Institute of Technology
Bangalore, India
sa.kavya@gmail.com

[2]Dept of Computer Science and Engineering
BTL Institute of Technology
Bangalore, India
raomvp@yahoo.com

[3]Dept of Computer Science and Engineering
BTL Institute of Technology
csehodbtlit@gmail.com

Abstract: ***In this paper, we addressed the problem of load balancing in large scale distributed systems. We study various load balancing strategies based on a tree representation of a Grid. The study allows transforming any Grid architecture into a unique tree with at most four levels. From this generic tree, we can derive three sub models depending on the elements that compose a Grid. Using this model, we defined a hierarchical load balancing strategy that privileges local balancing in first (load balance within groups without communication between groups). After load balancing at group level (if load is not balanced at group level) it will be balanced at region level and after balancing at region level (if load is not balanced at region level) it will be balanced at grid level.***

Keywords: Dynamic Load Balancing, Static Load Balancing.

1. Introduction

Grid computing is a form of networking. Unlike conventional networks that focus on communication among devices, grid computing harnesses unused processing cycles of all computers in a network for solving problems too intensive for any stand-alone machine. Grid computing requires the use of software that can divide and farm out pieces of a program to as many as several thousand computers. Grid computing can be thought of as distributed and large-scale cluster computing and as a form of network distributed parallel processing. It can be confined to the network of computer workstations within a corporation or it can be a public collaboration (in which case it is also sometimes known as a form of peer-to-peer computing).A number of corporations, professional groups, university consortiums, and other groups have developed or are developing frameworks and software for managing grid computing projects.

The European Community (EU) is sponsoring a project for a grid for high-energy physics, earth observation, and biology applications. In the United States, the National Technology Grid is prototyping a computational grid for infrastructure and an access grid for people. Sun Microsystems offers Grid Engine software. Described as a distributed resource management (DRM) tool, Grid Engine allows engineers at companies like Sony and Synopsys to pool the computer cycles on up to 80 workstations at a time. (At this scale, grid computing can be seen as a more extreme case of load balancing).

Grid computing appears to be a promising trend for three reasons: (1) its ability to make more cost-effective use of a given amount of computer resources, (2) as a way to solve problems that can't be approached without an enormous amount of computing power, and (3) because it suggests that the resources of many computers can be cooperatively and perhaps synergistically harnessed and managed as a collaboration toward a common objective. In some grid computing systems, the computers may collaborate rather than being directed by one managing computer. One likely area for the use of grid computing will be pervasive computing applications - those in which computers pervade our environment without our necessary awareness.

Thus grid computing is widely used to solve large-scale computational problems. Unlike traditional cluster computing, computational capabilities of resources in grid computing environments are usually different. In order to fulfill the user expectations in terms of performance and efficiency, the Grid system needs efficient load balancing algorithms for the distribution of tasks. A load balancing algorithm attempts to improve the response time of user's submitted applications by ensuring maximal utilization of available resources. The main goal is to prevent, if possible, the condition where some processors are overloaded with a set of tasks while others are lightly loaded or even idle[4]. Although load balancing problem in conventional distributed systems has been intensively studied, new challenges in Grid computing still make it an interesting topic and many research projects are under way. This is due to the characteristics of Grid computing and the complex nature of the problem itself. Load balancing algorithms in classical distributed systems, which usually run on homogeneous and dedicated resources, cannot work well in the Grid architectures [5].

Grids has a lot of specific characteristics, like heterogeneity, autonomy, scalability, adaptability and resources computation-data separation, which make the load balancing problem more difficult[6]. First we propose a dynamic tree-based model to represent Grid architecture in order to manage workload. This model is characterized by three main features: (i) it is hierarchical; (ii) it supports heterogeneity and scalability; and (iii) it is totally independent from any Grid physical architecture. Second, we develop a hierarchical load balancing strategy and associated algorithms based on neighborhood propriety. The goal of this idea is to decrease the amount of messages exchanged between Grid resources. As consequence, the communication overhead induced by tasks transferring and flow information is reduced.

2. Background Work

The structure of the Grid comprises characteristics of homogeneous as well as heterogeneous systems, loosely coupled as well as tightly coupled systems. Load balancing strategies aim to adapt the load optimally to the environment. However, they mainly consider the application running on a parallel, homogeneous system. Only a few methods address also the special characteristics of the underlying system. Zaki et al. [10] consider different processor speeds and distribute the load adequately. However, processor speeds are obtained by a prolong run and they assume full connectivity among the processors, with uniform latency and bandwidth. Hendrickson and Devine [4] review the major classes of dynamic load balancing (DLB) approaches. They point out that for heterogeneous systems, different amounts of computing power and memory should be considered. Additionally, they emphasis that network connections with different speeds are important for a DLB strategy. However, a solution is not proposed. Kielmann et al. [5] emphasis that a collection of clusters can be seen as a hierarchical system. They use a tree topology to do load balancing for divide-and-conquer applications. However, they do not take different PE characteristics into account. Willebeek and Reeves present in [9] a hierarchical balancing method (HBM). HBM is an asynchronous, global approach which organizes the system into a hierarchy of subsystems. The strategy has been implemented on a hypercube system. Due to its hierarchical structure, it is not necessary to perform any analysis of the topology at the beginning as well as modifications during runtime. Especially [5] and [9] show the importance of considering the underlying network. We did not found any methods in the literature that detect the hierarchical structure of a Grid environment and use the gained results to optimize middleware, as e.g. load balancing, specifically for this structure.

Min-Min, Max-Min and Sufferage algorithms are conventional scheduling algorithms and widely used in batch mode scheduling [5][6]. The details of these algorithms are as follows:

Min-Min: In the Min-Min scheduling algorithm, the task with lower computation time has higher priority. And, the task is assigned to the computing node which can finish executing it first.

Max-Min: Max-Min is the same as Min-Min in that the task is assigned to the computing node which can finish executing it first. But the difference is the task needing more computation time has higher priority.

Sufferage: The priorities of tasks in the Sufferage scheduling algorithm are given according to the sufferage value. This value is determined by the difference in computation time between the best and second best computing nodes. Then, the same as the above algorithms, tasks are assigned to the computing nodes which can finish them first.

Although the above three scheduling algorithm performed better than the traditional random or sequential scheduling approach, they ignore the importance of a dynamic network status. Usually, each task in a batch has a different computation and transmission time. The transmission time includes both transmitting the task to the computing node and returning the execution result after the task is finished. Because of the above factors, the total time for tasks will be effected by not only the computational capabilities of computing nodes but also the network status between these computing resources. However, not only above three scheduling algorithms but also some studies only take the computational capabilities into consideration and ignore the importance of the network status [4][11].

The Genetic algorithm (GA) was developed by Holland [8]. It simulates the evolution of natural biology which is based on Darwinian principles of natural selection. The GA operates on a population of chromosomes which is encoded according to the problem. Each chromosome in the population has a potential solution from the search space. With each generation, the chromosomes are operated by the reproduction, crossover, and mutation operators. Through these operators, not only the superior solutions can be preserved, but also an improved solution may be generated. Because of the above advantages, GA is widely used to solve heuristic problems by many researchers [1][10]. Many researchers have investigated the use of GA in homogeneous [2] and heterogeneous environments [7]. However, grid computing is a heterogeneous environment, so the technique for a homogeneous environment is not suitable. Although [7] proposed a GA based scheduling algorithm for grid computing environments, the crossover and mutation operators are controlled only by the fixed number of generations. In the evolutional phase, it is hard to predict if the fitness value is local optimization. So, controlling probabilities by a fixed number of generations is not suitable. In the worst case, if the mutation probability is gained with the fitness value not being convergent, the chance that the fitness value increasing may be lost.

3. PROBLEM DEFINITION

3.1 Existing system:

Load Balancing types:

A typical distributed system will have a number of interconnected resources who can work independently or in cooperation with each other. Each resource has owner workload, which represents an amount of work to be performed and every one may have a different processing capability. To minimize the time needed to perform all tasks, the workload has to be evenly distributed over all resources based on their processing speed. The essential objective of a load balancing consists primarily in optimizing the average response time of applications, which often means maintaining the workload proportionally equivalent on the whole resources of a system. Conceptually, load balancing algorithms [3] can be classified into two categories: static or dynamic [7].

The static load balancing problem for a mesh based application involves partitioning into sub domains. The sub domains can then be distributed over the processors and calculation carried out in parallel. Different partitions may result in different times to completion for the calculation. It is therefore necessary to examine the quality of the partitioning based on its effect on the application code. There are a number of factors.

The computational work of each processor should be balanced, so that no processor will be waiting for others to complete. Assuming that the computational work per processor is proportional to the number of mesh nodes in the sub domain, and then to achieve load balance it is necessary for the number of nodes in each sub domain to be the same.

When forming the discredited equations on a node of the mesh, the contributions from its nearest neighbor nodes will usually be needed. Depending on the order of the discretization scheme, contributions from more distant neighboring nodes may be necessary. On a parallel computer, accumulating the contributions from nodes that are not on the current processor will incur communication cost. It is known that on distributed memory parallel computers the cost of accessing remote memory is far higher than that of accessing local memory (typically a ratio of between 10 to 1000). It is therefore important to minimize the communication cost.

Dynamic Load Balancing (DLB) is used to provide application level load balancing for individual parallel jobs. It ensures that all loads submitted through the DLB environment are distributed in such a way that the overall load in the system is balanced and application programs get maximum benefit from available resources. Current version of the DLB has two major parts. One is called System Agent that collects system related information such as load of the system and the communication latency between computers. The other is called DLB Agent which is responsible to perform the load balancing. System Agent has to run all configured machine on the environment whereas DLB Agent is started by the user. Major components of the DLB are System Agent, and DLB Agent. Both components are written with Java. System requirements for DLB are LINUX/UNIX Operating Systems, Java 1.4 for System Agent (recommended) and DLB

- In static load balancing, a task is assigned to an available resource when it is generated or admitted to the system using a fixed schema.
- In contrast to static load balancing, dynamic load balancing allocate/reallocate tasks to resources at runtime based on no priori task information, which may determine when and whose tasks can be migrated. In this way, imbalances load can be resolved by redistributing tasks in real-time, thus solving the shortcoming of static load balancing. However, network traffic for transmitting load information to the load balancing system would increase too much due to the decision dynamicity. Load balancing algorithms can be defined by their implementation of the following policies [8]:
- Information policy: specifies what load information to be collected, when it is to be collected and from where.
- Triggering policy: determines the appropriate moment to start a load balancing operation. Resource type policy: classifies a resource as server or receiver of tasks according to its availability status and capabilities.
- Location policy: uses results of the resource type policy to find a suitable partner for a server or receiver.
- Selection policy: defines tasks that should be migrated from overloaded resources to idlest ones.

Load Balancing Problems:

Although load balancing methods in conventional parallel and distributed systems has been intensively studied [4], they do not work in Grid architectures because these two classes of environments are radically distinct. Indeed, the schedule of tasks on multiprocessors or multi computers suppose that processors are homogeneous and linked with homogeneous and fast networks[9]. The rationale behind this approach is that:

1. The resources have same capabilities;
2. The interconnection bandwidth between processing elements is high;
3. Input data is readily available at the processing site;
4. The overall time spent transferring input and output data is negligible in comparison with the total application duration. Given the distribution of tremendous resources in a Grid environment and the size of the data to be moved, it becomes clear that this approach is not accurate because following properties [5,6].

A. Heterogeneity exists in both of computational and networks resources.

- First, networks used in Grids may differ significantly in terms of their bandwidth and communication protocols.
- Second, computational resources are usually heterogeneous (processors, resource capabilities memory size and so on). Also different software's, like operating systems, file systems; cluster management software may be heterogeneous.

B. Autonomy: Because the multiple administrative domains that share Grid resources, a site is viewed as an autonomous computational entity. It usually has its own scheduling policy, which complicates the task allocation problem. A single overall performance goal is not feasible for a Grid system since each site has its own performance goal and scheduling decision is made independently of other sites according to its own performances.

C. Scalability and adaptability: A Grid might grow from few resources to millions. This raises the problem of potential performance degradation as the size of a Grid increases. If the pool of resources can be assumed fixed or stable in traditional parallel and distributed computing environments, in Grid dynamicity exists in the networks and computational resources.

- First, a network shared by many execution domains cannot provide guaranteed bandwidth.
  This is particularly true for Wide-Area Networks like Internet.
- Second, both the availability and capability of computational resources will exhibit dynamic behavior. On one hand new resources may join the Grid and on the other hand, some resources may become unavailable. Resource managers must tailor their behavior dynamically so that they can extract the maximum performance from the available resources and services.

D. Resource selection and computation: Data separation: In traditional systems, executable codes of applications and input/output data are usually in the same site, or the input sources and output destinations are determined before the submission of an application. Thus the cost for data staging can be neglected or the cost is a constant determined before execution and load balancing algorithms need not consider it. But in a Grid the computation sites of an application are usually selected by the Grid scheduler according to resource status and some performance criterion. Additionally, the communication bandwidth of the underlying network is limited and shared by a host of background loads, so the communication cost cannot be neglected. This situation brings about the computation-data separation problem: the advantage brought by selecting a computational resource that can provide low computational cost may be neutralized by its high access cost to the storage site. These challenges pose significant obstacles on the problem of designing an efficient and effective load balancing system for Grid environments. Some problems resulting from the above are not solved successfully yet and still open research issues. Thus it is very difficult to define a load balancing system which can integrate all these factors.

3.2 Proposed system:

Tree-Based Balancing Model

In order to well explain our model, we first define the topological structure for a grid computing.

Grid topology:

We suppose that a grid computing (see Fig. 1) is a finite set of G clusters C_k , interconnected by gates gt_k , $k \in \{0, ..., G - 1\}$, where each cluster contains one or more sites S_{jk} interconnected by switches SW_{jk} and every site contains some Computing Elements CE_{ijk} and some Storage Elements SE_{ijk} , interconnected by a local area network.

Load balancing generic model:

- Level 0: In this first level (top level of the tree), we have a virtual node that corresponds to the root of the tree. It is associated to the grid and performs two main functions: (i) manage the workload information of the grid; (ii) decides, upon receiving tasks from users, where these tasks can be launched, based on the user requirements and the current load of the grid.
- Level 1: This level contains G virtual nodes, each one associated to a physical cluster of the grid. In our load balancing strategy, this virtual node is responsible to manage workload of its sites.
- Level 2: In this third level, we find S nodes associated to physical sites of all clusters of the grid. The main function of these nodes is to manage the workload of their physical computing elements.
- Level 3: At this last level (leaves of the tree), we find the M Computing Elements of the grid linked to their respective sites and clusters.

Figure 2 shows the generic tree model associated to a grid, with its three variants: 1/1/M, 1/S/M and G/S/M.

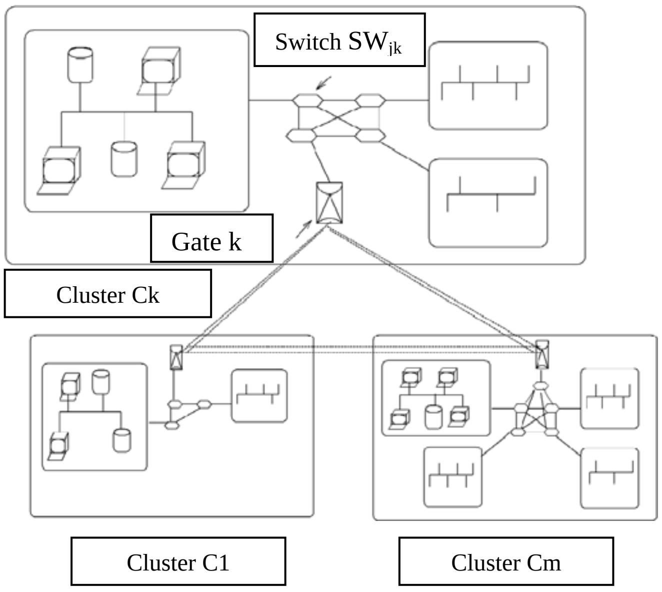

Figure 1: Example of Grid Computing

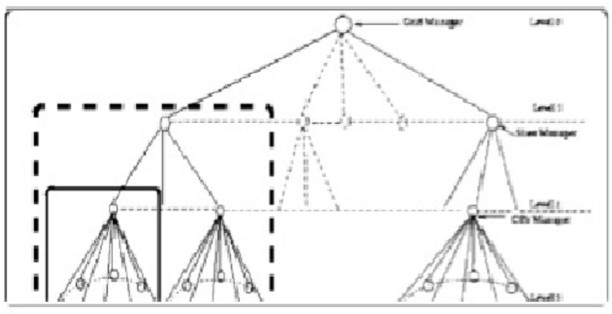

Figure 2: Tree-based representation of a grid

Characteristics of the proposed model:

The main features of our proposed load balancing generic model are listed below:

- It is hierarchical: this characteristic facilitate the in-formation flow through the tree and well defines the message traffic in our strategy.
- It supports heterogeneity and scalability of grids: adding or removing entities (computing elements, sites or clusters) are very simple operations in our

model (adding or removing nodes, subtrees).

- It is totally independent from any physical architecture of a grid: the transformation of a grid into a tree is an univocal transformation. Each grid corresponds to one and only one tree.

LOAD BALANCING STRATEGIES:

A. Intra-group load balancing: In this first level, depending on its current load, each node manager decides to start a load balancing operation. In this case, the node manager tries in priority, to load balance its workload among its computing elements.

B. Intra-region load balancing: In this second level, load balance concerns region, for which some owner node managers fail to achieve a local load balance. In this case, the group manager transfers tasks from overloaded groups to under loaded ones.

C. Intra-Grid load balancing: The load balance at this level is used only if some group managers fail to load balance their workload among their associated groups. If we have such as case, tasks of overloaded regions are transferred to under loaded regions by the Grid manager. The main advantage of this strategy is to privilege local load balancing in first (within a group, then within a region and finally on the whole Grid). The goal of this neighborhood strategy is to decrease the amount of messages between groups and regions. As consequence of this goal, the communication overhead induced by tasks transfer is reduced.

D. Generic strategy: At any load balancing level, we propose the following strategy:

1. Estimate the current workload of a group, a region or a Grid: Here we are interested by the information policy to define what information reflects the workload status of group/region/Grid, when it is to be collected and from where. Knowing the number of available elements under his control and their computing capabilities, each group manager estimates its own capability and performs the following actions:

i. Estimates current group workload based on workload information received periodically from its elements.

ii. Computes the standard deviation over the workload index in order to measure the deviations between its involved elements.

iii. Workload information to its manager.

2. Decision-making: In this step the manager decides whether it is necessary to perform a load balancing operation or not. For this purpose it executes the two following actions:

i. Determines the imbalance/saturation state. If we consider that the standard deviation measures the average deviation between the processing time of an element and the processing time of its group (group/region/Grid), we can say that this element is in balance state when this deviation is small. Indeed this implies that processing time of each element converges to the processing time of its group. An element can be balanced while being saturated. When the current workload of the element cross its capacity, it is obvious that it is useless to balance since all belonging components are saturated.

ii. Partitioning. For an imbalance case, we determine the overloaded elements (sources) and the under loaded ones (receivers), depending on processing time of every element relatively to average processing time of the associated group.

3. Tasks transfer: In order to transfer tasks from overloaded elements to under loaded ones, we propose the following heuristic:

i. Evaluate the total amount of load:"Supply", available on receiver elements.

ii. Compute the total amount of load:"Demand", required by source elements.

iii. If the supply is much lower than the demand (supply is far to satisfying the request) it is not recommended to start local load balancing. We introduce a third threshold, called expectation threshold , to measure relative deviation between supply and demand.

iv. Otherwise performs tasks transfer regarding communication cost induced by this transfer and according to criteria selection.

4. EXPERIMENTAL STUDY

Modeling parameters:

In order to evaluate the practicability and the performance of our model we have developed a grid simulator. This simulator was built in Java and uses the following parameters:

1) CE's parameters: these parameters give information about available CE's during load balancing period such as: (i) number of sites; (ii) number of CE's in each site; (iii) CE's speeds; (iv) date to send workload information from CE's; and, (v) tolerance factor.

2) Tasks parameters: these parameters include: (i) number of tasks queued at every CE; (ii) task submission date; (iii) number of instructions per task; (iv) task size; and, (v) priority.

3) Network parameter: bandwidth size.

4) Workload index: as workload for Computing Elements, we have used their occupation ratio: workload=inst/speed, where inst denotes the total number of instructions queued on a given CE and speed is its speed.

5) Performance parameters: in our experimentations, we focused on two performance parameters: tasks average response time and cost communication.

Experimental results:

All the experiments were performed on PC Pentium IV of 2.8 GHz, with a 256 MB RAM and running under Windows XP. In order to obtain reliable results, we reiterated the same experiments more than ten (10) times.

In the sequel, we will give the experimental results relating to the response time according to the number of tasks and according to the number of computing elements. The following tables (see Tables I and II) show the variation of the average response time before and after execution of the intra-site load balancing algorithm.

In Tables I and II, Before and After represent mean response time before and after load balancing is performed and cost defines the communication cost expressed in seconds. From these tables, we remark that our strategy leads to a good load balancing:

1) For a number of tasks fixed at 2000 and for a number of CE's varying from 50 to 250 by step of 50, we obtain a gain varying between 10.65% and 21.43%, with negligible cost of communication.

2) For a number of CE's equal to 250 and for a number of tasks varying from 1000 to 2000 by step of 250, the gain varies from 14.93% to 15.93%.

3) During our experiments, we have remarked that the best gains are obtained when the grid is in a stable state (neither overloaded nor completely idle).

Table1: RESPONSE TIME VS NUMBER OF CE'S (NUMBER OF TASKS=2000)

#CE's	**Response time (sec)**			**Cost**
	Before	**After**	**Gain(%)**	**(sec)**
50	817	730	10.65	41
100	409	353	13.69	50
150	273	230	15.75	49
200	126	99	21.43	47
250	164	139	15.24	49

Table2: RESPONSE TIME VS NUMBER OF TASKS (NUMBER OF CE'S=250)

#CE's	**Response time (sec)**			**Cost (sec)**
	Before	**After**	**Gain (%)**	
1000	113	95	15.93	22
1250	134	114	14.93	33
1500	145	145	15.86	37
1750	156	156	15.38	47
2000	164	164	15.24	49

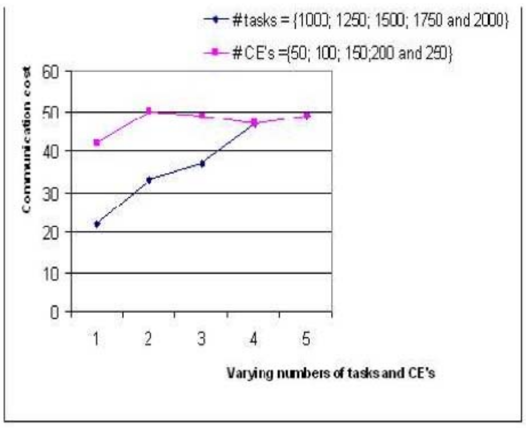

Figure 3: Communication time Vs Number of CE's and tasks

5. Conclusion and Future works

In this paper, we addressed the problem of load balancing in grid computing. We proposed a load balancing strategy based on a tree model representation of grid architecture. The model allows the transformation of any grid architecture into a unique tree with at most four levels. From this generic tree, we can derive three sub-models depending on the elements that compose a grid: one site, one cluster, or in the general case multiple clusters. Using this model, we defined a hierarchical load balancing strategy that gives priority to local load balancing within sites The proposed strategy leads to a layered algorithm which an prototype was implemented and evaluated on a grid simulator developed for the circumstance.The first results of our experimentations show that the proposed model can lead to a better load balancing between CE's of a grid without high overhead. We have observed that significant benefit in mean response time was realized with a reduction of communication cost between clusters.

The model presented in this paper raises a number of challenges for further researches. First, we plan to test our model on others grid simulators [11]. Second, we plan to experiment our model on real grid environments like Globus [13] and XtremWeb [12], using a realistic grid application, in order to validate the practicality of the model.

References

[1] Buyya, R., D. Abramson, J. Giddy and H. Stockinger, 2002. Economic models for resource management and scheduling in grid computing. J. Concurrency and Computation: Practice and Experience, 14: 1507-1542.

[2] Foster, I., C. Kesselman and S. Tuecke, 2002. The anatomy of the Grid: Enabling scalable virtual organizations. Intl. J. High Performance Computing Applications, 15: 3.

[3] Xu, C.Z. and F.C.M. Lau, 1997. Load Balancing in Parallel Computers: Theory and Practice. Kluwer, Boston, MA.

[4] Berman, F., G. Fox and Y. Hey, 2003. Grid Computing: Making the Global Infrastructure a Reality. Wiley Series in Comm. Networking & Distributed System.

[5] Zhu, Y., 2003. A survey on grid scheduling systems. Technical report, Department of Computer Science, Hong

Kong University of science and Technology.
[6] Houle, M., A. Symnovis and D. Wood, 2002. Dimension-exchange algorithms for load balancing on trees. In Proc. of 9th Int. Colloquium on Structural Information and Communication Complexity, pp: 181-196.
[7] Kabalan, K.Y., W.W. Smar and J.Y. Hakimian, 2002. Adaptive load sharing in heterogeneous systems: Policies, modifications and simulation. Intl. J. Simulation, 3: 89-100.
[8] Leinberger, W., G. Karypis, V. Kumar and R. Biswas, 2000. Load balancing across nearhomogeneous multi-resource servers. In 9th Heterogeneous Computing Workshop, pp: 60-71.
[9] Buyya, R., A Grid simulation toolkit for resource modelling and application scheduling for parallel and distributed computing. www.buyya.com/gridsim/.
[10] Foster, I. and C. Kesselman, 1997. Globus: a metacomputing infrastructure toolkit. Intl. J. Super-Computer and High Performance Computing Applications, 11: 115-128.
[11] GridSim. A grid simulation toolkit for resource modeling and application scheduling for parallel and distributed computing. www.buyya.com/gridsim/.
[12] XtremWeb.A global computing experimental platform.http://www.lri.fr/fedak/XtremWeb/introduction.ph
[13] I.Foster and C.Kesselman. Globus: a metacomputing infrastructure toolkit. Int. Jour. of Super-Computer and High Performance Computing Applications, 11(2):115-128, 1997.

Author Profiles

Kavya.S.A received the B.E. degree in Computer Science and Engineering from BTL Institute of Technology, Bangalore. At present persuing the Master of Technology in Computer Science and Engineering Department at BTL institute of Technology, Bangalore.

M.V.Panduranga Rao is a Researcher at NITK, India. He received the M.Tech degree in computer Science from Visvesvaraya Technological University and B.E. degree in Electronics and Communication from Kuvempu University, Karnataka. He was Research Associate at JNNCE, during the period from August 1989 – April 2005. He received an award in Okinawa, Japan.

S Basavaraj Patil is the Founder & Principal Consultant; Predictive Research He received the PhD in, Computer Science & Engineering from Kuvempu Vishwavidyanilaya and M.Tech, Bio-Medical Instrumentation from S J College of Engineering, Mysore. He worked as Assistant Vice President at CIBM Research, and HSBC Consultant at Manthan Systems and Technical Architect at Aris Global. He is presently working as head of the department at BTL institute of technology.